**­­­­Министерство науки и высшего образования Российской Федерации**

Федеральное государственное автономное образовательное учреждение

высшего образования «Уральский федеральный университет

имени первого Президента России Б.Н. Ельцина»

Институт радиоэлектроники и информационных технологий - РТФ

Центр ускоренного обучения

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №6**

по дисциплине «Конструирование программного обеспечения»

**Тема:** **Паттерны проектирования**

Студенты гр. РИВ-400027у Л.А.Кайгородова

О.В.Дрон

Д.И.Кудинов

В.И.Пинтак

Я.В.Козлов

Преподаватель С.И.Тимошенко,

доц., к.т.н

**Екатеринбург 2023**

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc146030453)

[2 Результаты проделанной работы 4](#_Toc146030454)

[2.1 Фабричный метод 4](#_Toc146030455)

[2.2 Адаптер 5](#_Toc146030456)

[2.3 Цепочка обязанностей 6](#_Toc146030457)

[3 Анализ полученных результатов 9](#_Toc146030458)

# 1 Постановка задачи

1. Изучить паттерны проектирования по книге Швец А. Погружение в паттерны проектирования / А. Швец // Сайт Sergey Drozdov. – [2022]. – Режим доступа: https://sd.blackball.lv/library/ Pogruzhenie\_v\_patterny\_proektirovanija (2021).pdf (дата обращения 01.07.2022). Книга есть также в папке “Книги\_и\_статьи\Паттерны (шаблоны) проектирования”.
2. Выбрать варианты паттернов для исследования, исходя из номера команды по таблице (вариант 1).
3. Ознакомиться с паттернами по указанной в п. 1 книге.
4. Загрузить в одну из IDE проект с паттернами на Java (лучше взять из папки “Швец А. Погружение в паттерны проектирования. 2021 - Java (проект в Eclipse)”).
5. Выполнить паттерны, убедившись в работоспособности кода и разобравшись в принципах его работы.
6. Выполнить обратное проектирование кода (reverse engineering), построив по коду диаграмму классов.
7. В отчет по лабораторной работе поместить полученные диаграммы классов для каждого паттерна и описать по ним принцип работы паттерна.

# 2 Результаты проделанной работы

Наша команда взяла первый вариант паттернов проектирования из таблицы: фабричный метод, адаптер, цепочка обязанностей. Далее будет рассмотрен каждый из данных паттернов.

## 2.1 Фабричный метод

Данный паттерн относится к порождающему типу, который определяет общий интерфейс для создания объектов в суперклассе, позволяя подклассам изменять тип создаваемых объектов. Для этого используется «фабричный» метод, который отвечает за создание объектов и часто бывает абстрактным, передавая возможность наследникам класса реализовать такой метод по-своему.

Проект с паттернами на Java был загружен в IDE Eclipse. Результат запуска паттерна «фабричный метод» представлен на рисунке 1.



Рисунок 1 – Результат выполнения паттерна «фабричный метод»

В зависимости от операционной системы результат работы данной программы будет различаться: для Windows 10 появляется окно диалога (как в данном случае), при нажатии на кнопку «Exit» окно закрывается. Для остальных операционных систем должен появиться HTML-диалог.

Выполним обратное проектирование кода для данного паттерна и разберёмся в принципах его работы. На рисунке 2 представлена диаграмма классов для паттерна «фабричный метод».

В нашем случае фабричным является метод createButton(): Button, который помогает создавать кросс-платформенные элементы интерфейса, не привязывая основной код программы к конкретным классам; он объявлен в абстрактном классе-фабрике Dialog. Тип возвращаемого значения у фабричного метода совпадает с интерфейсом Button. Наследниками класса Dialog являются классы – конкретные фабрики: WindowsDialog и HtmlDialog, которые реализуют фабричный метод, и в зависимости от операционной системы мы получаем различный результат (собственный продукт) диалогового окна. Button является общим интерфейсом для классов WindowsButton и HtmlButton, которые, в свою очередь, реализуют его методы для отрисовки и обработки событий – render(): void и onClick(): void. Поэтому в классе Demo, в котором находится главный метод main(args: String[]): void, создаётся конкретный продукт – диалоговое окно класса-фабрики в зависимости от окружения.

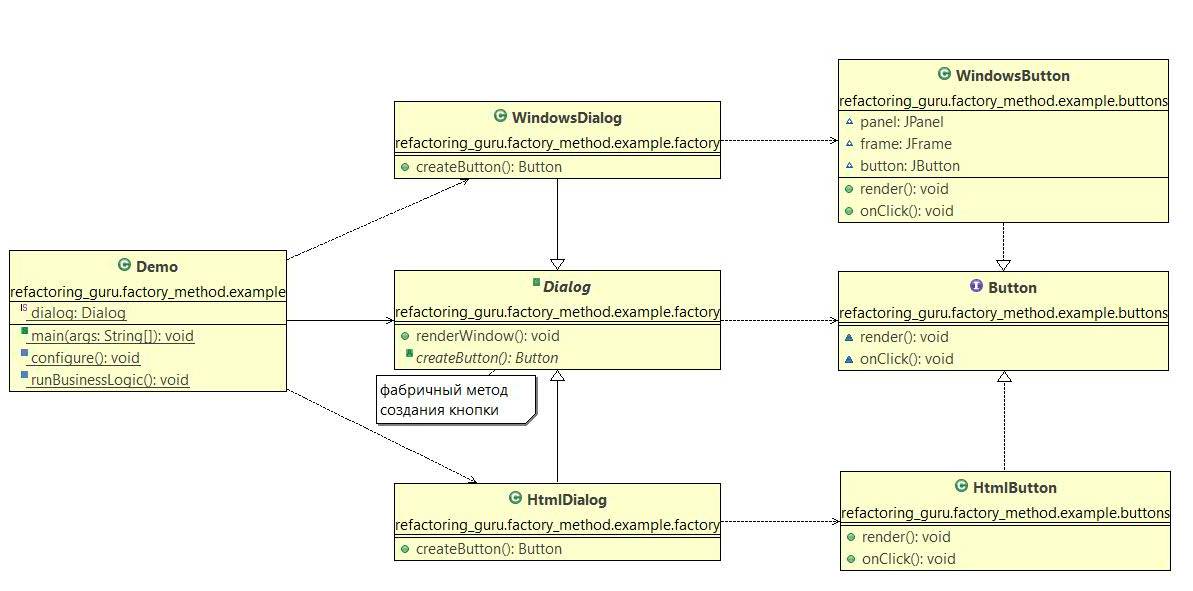


Рисунок 2 – Диаграмма классов паттерна «фабричный метод»

## 2.2 Адаптер

Адаптер является структурным паттерном проектирования, который позволяет объектам с несовместимыми интерфейсами работать вместе. Для этого создаётся отдельный класс-адаптер, который трансформирует данные объекта класса-сервиса в такой вид, чтобы он стал понятен другому объекту класса-клиента.

После запуска данного паттерна мы получим консольный вывод (рисунок 3).

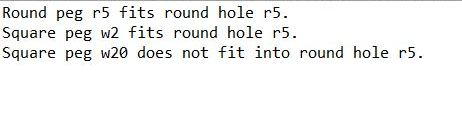


Рисунок 3 – Результат выполнения паттерна «адаптер»

В данной программе класс-адаптер рассчитывает радиус квадратного колышка исходя из его ширины, чтобы потом сравнить его с радиусом круглого отверстия, предназначенного для круглого колышка. В текущем выводе в консоль появляется сообщение о том, что круглый колышек с радиусом 5 и квадратный колышек с шириной 2 подходят для отверстия с радиусом 5, а квадратный колышек с шириной 20 не подходит для данного отверстия.

Выполним обратное проектирование кода и разберёмся в принципах его работы. На рисунке 4 представлена диаграмма классов для паттерна «адаптер».

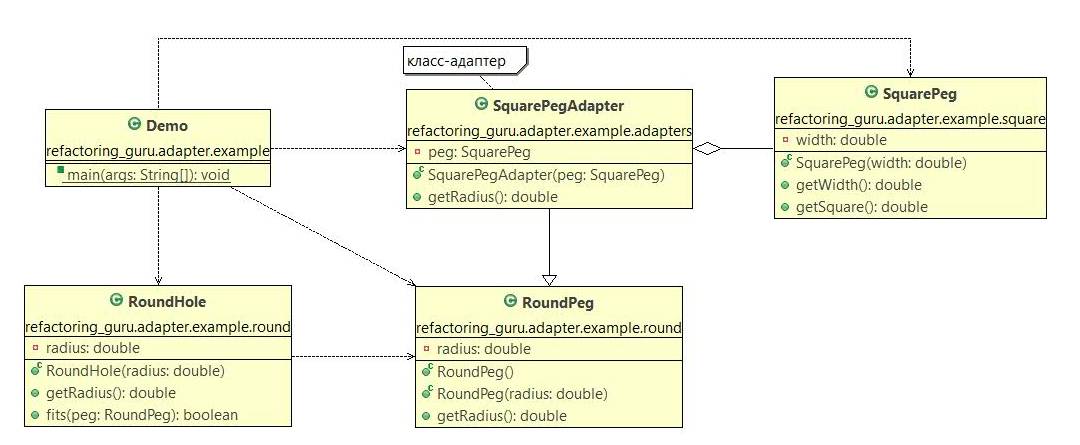


Рисунок 4 – Диаграмма классов паттерна «адаптер»

Классом-адаптером в данном случае является класс SquarePegAdapter, который в методе getRadius(): double рассчитывает радиус квадратного колышка(объекта класса-сервиса SqarePeg), переопределяя данный метод у класса-родителя RoundPeg, который возвращает радиус круглого колышка. Также класс-адаптер содержит в себе поле-ссылку на объект класса SquarePeg, поэтому классы SquarePegAdapter и RoundPeg находятся в отношении агрегации.

В классе Demo содержится бизнес-логика приложения: в методе main(args: String[]): void создаются объект-круглое отверстие класса RoundHole(класс-клиент) и объект-круглый колышек класса RoundPeg и сравниваются их радиусы. Затем создаётся объект класса-адаптера, в конструктор которого передаётся объект-квадратный колышек класса SquarePeg и сравнивается с круглым отверстием, после чего в консоль выводится результат сравнения.

## 2.3 Цепочка обязанностей

Цепочка обязанностей – это поведенческий паттерн проектирования, который позволяет передавать запросы последовательно по цепочке объектов-обработчиков с единственным методом, куда передаются данные. Каждый последующий обработчик решает, может ли он обработать запрос сам и стоит ли передавать запрос дальше по цепи. Каждый из объектов имеет ссылку на следующий обработчик цепи.

Запустим данный паттерн в Eclipse. Нам будет предложено ввести email и пароль, и в зависимости от введённых нами данных в случае успешной авторизации мы получим различный консольный вывод для администратора и пользователей (рисунок 5, 6).



Рисунок 5 – Результат выполнения паттерна «цепочка обязанностей» для администратора



Рисунок 6 – Результат выполнения паттерна «цепочка обязанностей» для пользователя

Если email или пароль был введён неверно, в консоль выводится сообщение об ошибке в данных, также мы не сможем войти в систему, если ввод данных займёт у нас более 60 секунд, в таком случае в консоль будет выведено сообщение о превышении лимита времени. Таким образом демонстрируется процесс авторизации в системе.

Выполним обратное проектирование кода и разберёмся в принципах его работы. На рисунке 7 представлена диаграмма классов для паттерна «цепочка обязанностей».

Абстрактный класс Middleware является базовым классом для всех обработчиков в данной программе. Он содержит абстрактный метод check(email: String, password: String): boolean, с помощью которого будет осуществляться проверка во время авторизации при его переопределении классами-наследниками. Класс Middleware также содержит поле-ссылку на объект класса-наследника, эта ссылка передаётся в метод linkWith(next: Middleware): Middleware, таким образом происходит связь между классами цепочки. Метод checkNext(email: String, password: String): boolean запускает следующего обработчика.

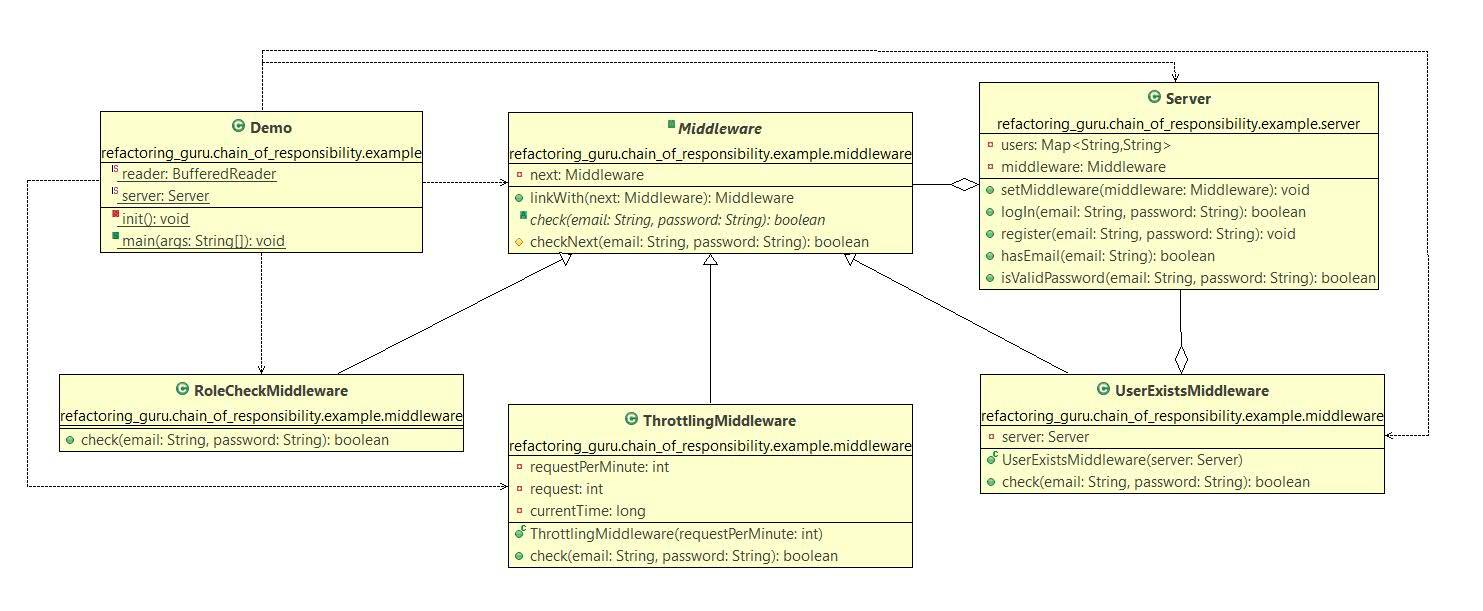


Рисунок 6 – Диаграмма классов для паттерна «цепочка обязанностей»

Наследниками базового класса цепи Middleware являются классы UserExistsMiddleware, ThrottlingMiddleware, RoleCheckMiddleware, каждый из них выполняет определённую проверку при авторизации, так они создают «цепочку обязанностей».

Класс Server получает email и пароль от клиента и запускает проверку авторизации у цепочки с помощью метода logIn(email: String, password: String): boolean. Для хранения email и пароля пользователя используется динамический массив users.

Класс UserExistsMiddleware содержит поле-ссылку на объект класса Server и с помощью переопределённого метода check(email: String, password: String): boolean проверяет, есть ли введённый пользователем email в массиве users. Внутри данного метода содержится проверка введённого пароля с помощью метода isValidPassword(email: String, password: String): boolean, который сравнивает введённый пароль с паролем из массива users.

Класс ThrottlingMiddleware переопределяет, в свою очередь, метод check(email: String, password: String): boolean таким образом, что происходит проверка допустимого времени для ввода данных – 60 секунд, по истечении которого выводится сообщение о превышении данного времени.

Класс RoleCheckMiddleware переопределяет метод check(email: String, password: String): boolean так, что при успешном вводе данных в консоль выводится сообщение с приветствием пользователя или администратора.

В классе Demo вся логика программы, распределённая на отдельные классы-обработчики, сводится воедино: создаётся цепочка проверок и передаётся серверу.

# 3 Анализ полученных результатов

В результате выполнения лабораторной работы № 6 были описаны следующие паттерны проектирования: фабричный метод, адаптер, цепочка обязанностей. Данные паттерны были выполнены на Java в IDE Eclipse. Затем было выполнено обратное проектирование кода и построены диаграммы классов.